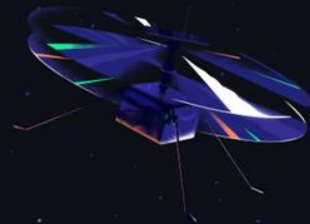




GitHub Copilot Training

Andrew Scoppa



AGENDA

GitHub Copilot - Introduction

Best practices & prompt engineering

In-class coding demos using copilot and copilot chat

Secure coding

Wrap-up, Q&A



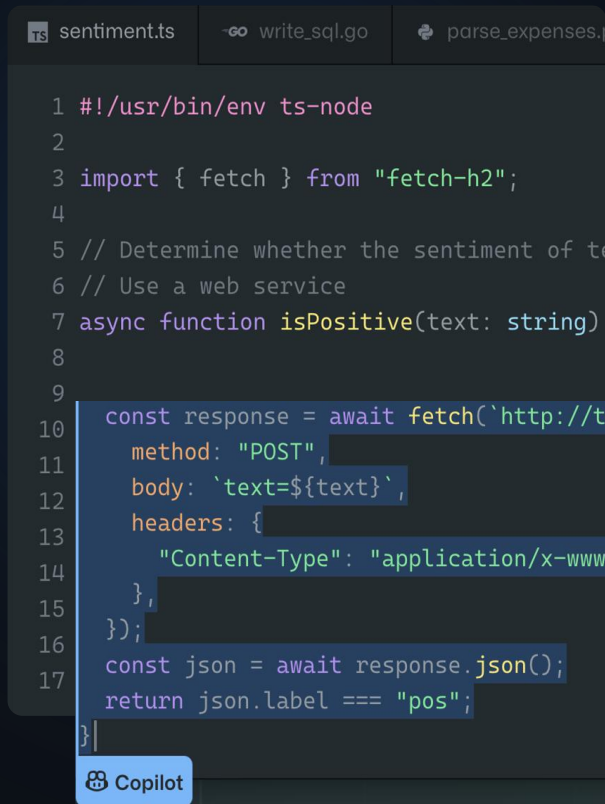


GitHub Copilot Introduction



Let's start with a high-level overview of GitHub Copilot


- GitHub Copilot is there to enhance daily work
 - Like a smart assistant or mentor by your side
- Draws context from text & code in open tabs
- Powered by OpenAI
 - Copilot uses a transformative model
 - Think of something like Google Translate
- Trained on large datasets to ensure accuracy
 - It even can help with HTML and markdown!
- Copilot generates new code in a probabilistic way, and the probability that it produces the same code as a snippet that occurred in training is low.



```

1  #!/usr/bin/env ts-node
2
3  import { fetch } from "fetch-h2";
4
5  // Determine whether the sentiment of text is positive
6  // Use a web service
7  async function isPositive(text: string): Promise<boolean> {
8
9
10     const response = await fetch(`http://localhost:3000/sentiment`, {
11       method: "POST",
12       body: `text=${text}`,
13       headers: {
14         "Content-Type": "application/x-www-form-urlencoded",
15       },
16     });
17     const json = await response.json();
18     return json.label === "pos";
19   }

```

 Copilot



Copilot vs Copilot Chat

Copilot

Direct Code Writing

Seamless IDE Integration

Solo Development

Copilot Chat

In-Depth Interactive Assistance

Provides slash commands ("/")

Collaborative Scenarios



Slash Commands (‘/’)

- Provide quick access to specific functionality
- Perform common actions quickly without typing full instructions
- Supported by built-in and extension-provided chat participants.

- `/explain` - Explain how the code in your active editor works
- `/tests` - Generate unit tests for the selected code
- `/fix` - Propose a fix for the problems in the selected code
- `/new` - Scaffold code for a new file or project in a workspace
- `/newNotebook` - Create a new Jupyter Notebook

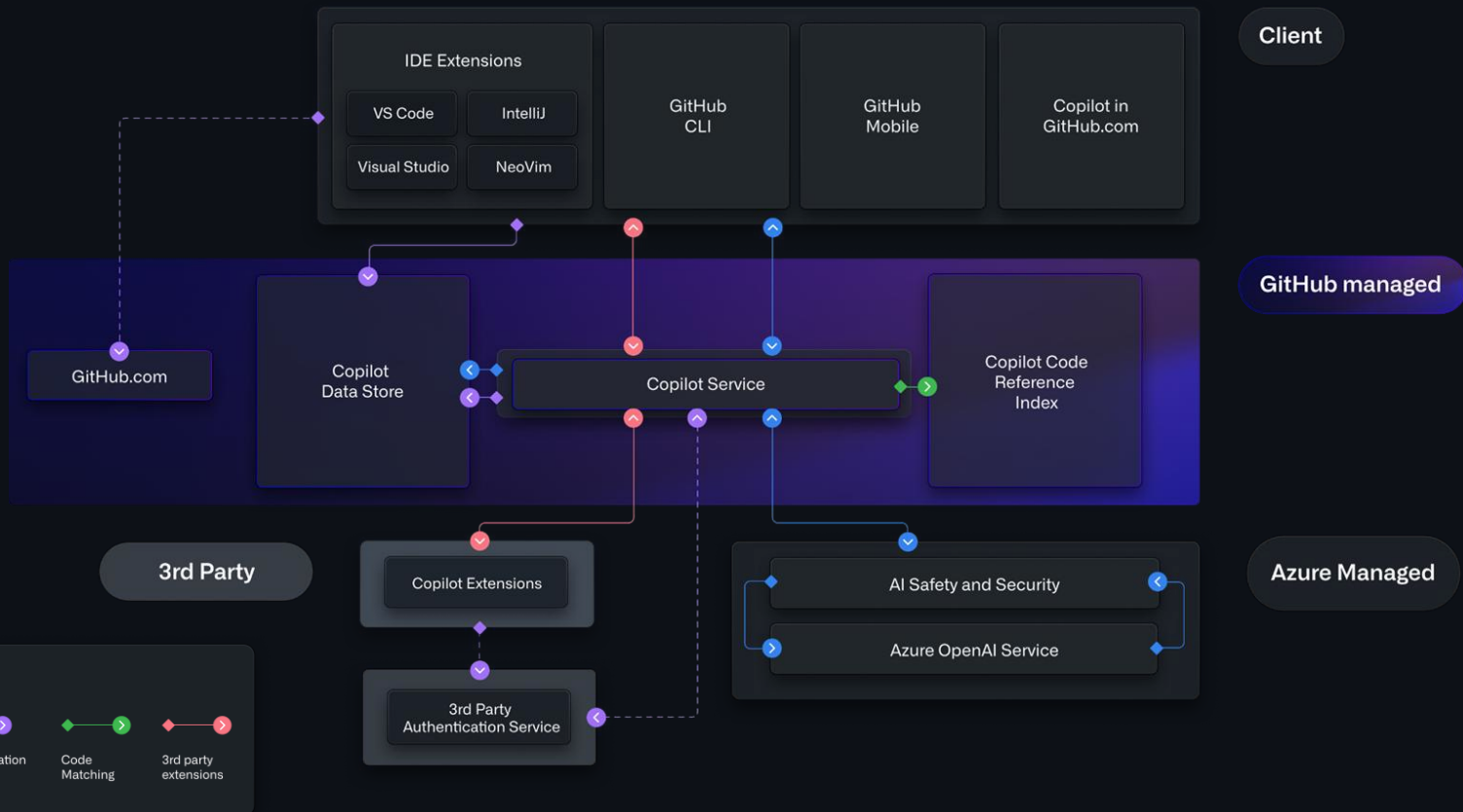




Helps developers stay in the flow throughout the entire SDLC



GitHub Copilot Architecture



GPT Models

Copilot is always evolving and much of this evolution is powered by the evolving models behind it



Past (Codex)

- AI Hallucination problem
- Outdated/Deprecated code
- Lack of contextual understanding
- Failing to suggest best practices
- But it worked!

Present (GPT-3.5 GPT-4)

- Faster
- More efficient in regards to GPU usage
- Larger context window
- More recent training data

Future (Specific Models)*

- Code-specific models
- Enhanced experience and quality through fine-tuning
- Customization of models

* GPT keeps evolving and the latest can be found in github.com/blog



Prompt Engineering



What is Prompt Engineering?



Prompt Engineering is the strategic crafting of user inputs to guide the AI towards producing desired code outputs.

A nuanced understanding of prompt engineering allows developers to optimize their use of Copilot, ensuring they get the most out of this advanced coding assistant.

```
15  /*
16  Create a function 'sqrGen' which returns a lamda function
17  that will square the argument passed to the lambda
18  Example usage:
19  auto sqr = sqrGen(); int result = sqr(5); // result will be 25
20  ✨
21  auto sqrGen() {
22  ... return [](int m) -> int {return m * m;};
23  }
```



AI Prompt Techniques

Zero Shot Prompt

```
1 //Write a JavaScript function that finds
2 //the maximum value in an array.
3 function max(array) {
4     return Math.max.apply(null, array);
5 }
```

One-Shot Prompt

```
1 ✓ //Create a JavaScript function, `findMax`, that takes an array of strings as input
2 //and returns the string with the maximum length.
3 //Ensure that the function works correctly for arrays that are empty,
4 //as well as arrays that contain both strings and numbers.
5 ✓ function findMax(array) {
6     var max = array[0].length;
7     array.map(v => max = Math.max(max, v.length));
8     result = array.filter(v => v.length == max);
9     return result;
10 }
```



Providing Context to Copilot

To help Copilot generate accurate suggestions:

- Add a top-level comment block describing the purpose of the file
- Front load as many imports as needed (import / include / requires / etc.)
- Create a detailed comment block describing the purpose of an operation or UDT
- Use sample code as a starting point
- Have related content open in other tabs



Helpful Patterns

- * Use descriptive variable names to make your intentions clear.

```
totalSampleCount = 1000
```

- * Maintain consistent naming conventions for variables and functions.

i.e. using camelCase for variable names consistency

- * Define method signatures with unambiguous parameter names and types.

```
double calculateAverageSampleSize(unsigned long samples[], size_t size)
```

- * Use comments to explain the purpose of the code.

```
This function 'palindrome' takes an array of strings  
as input and returns true if a palindrome is found.
```



Helpful Patterns

- * Specify error handling scenarios.

Exit where the input is NULL and throw an exception if the input out of range [min, max] inclusive.

- * Describe control flow structures.

Write a while loop to print the first 'n' values in the fibonacci sequence.

- * Show examples of how to use the code.

```
int samples[] = {1, 2, 3, 4, 5};  
double average = calculateAverageSampleSize(samples, 5);  
printf("Average: %f\n", average); // Average: 3.0
```

- * Test your code.

Write a test suite to verify the correctness of the code.

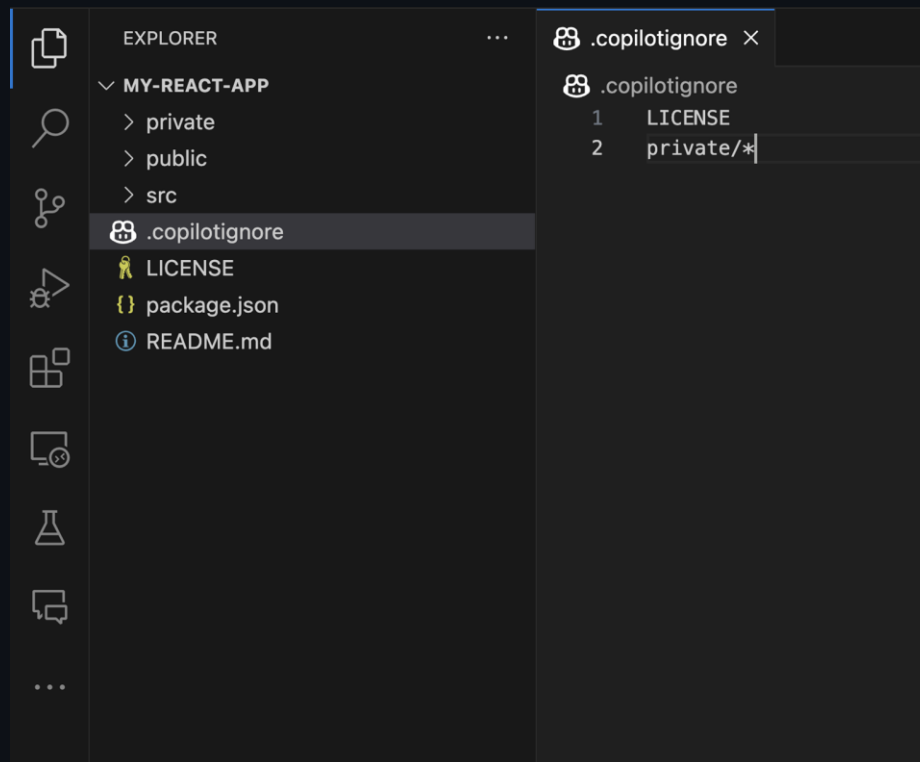


Secure coding



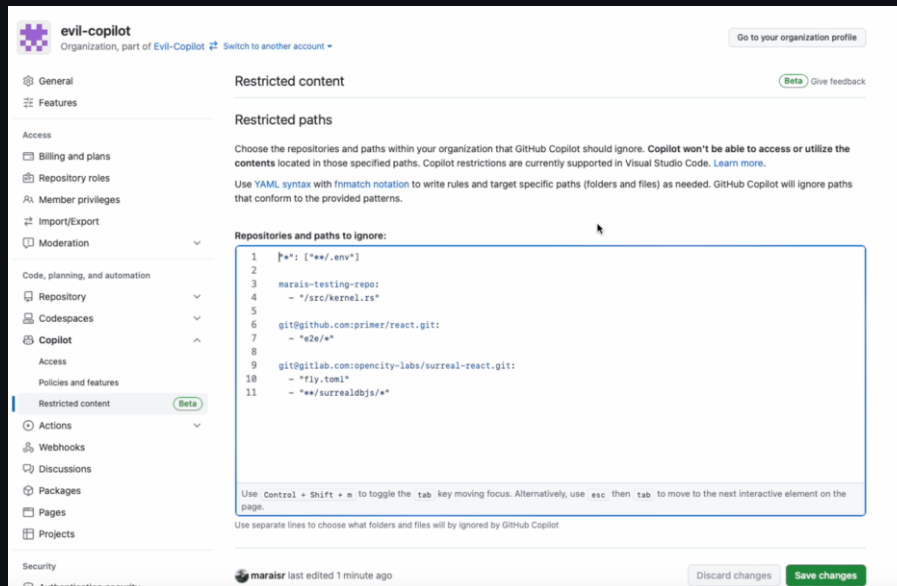
Block files from Copilot

Use `.copilotignore` to block files and folders from being used by Github Copilot



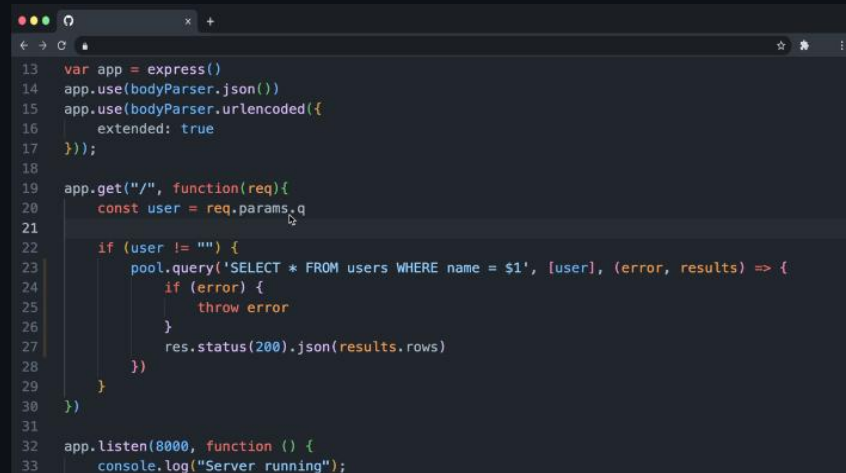
Block files from Copilot

- The Copilot Restricted Content feature allows repository owners to control the usage of Copilot across their codebases.
- This includes specific files, folders, or entire repositories, even if they're not hosted on GitHub.
- The goal is to provide CfB customers a flexible way of controlling what content Copilot can access for prompt crafting and where it can insert code.



Copilot and Secure Coding

- AI-based vulnerability system that helps prevent insecure coding patterns (e.g. SQL script injection)
- Vulnerability filters are applied only to the Copilot-generated suggestion itself
 - ⊗ It cannot detect downstream vulnerabilities introduced by the code e.g. on deployment infrastructure
 - ⓘ **We recommend taking the same precautions you take with code written by engineers (linting, code scanning, etc.)**
- Copilot Chat can be used to query code for known vulnerabilities



```

13 var app = express()
14 app.use(bodyParser.json())
15 app.use(bodyParser.urlencoded({
16   extended: true
17 }));
18
19 app.get("/", function(req){
20   const user = req.params.q
21
22   if (user != "") {
23     pool.query('SELECT * FROM users WHERE name = $1', [user], (error, results) => {
24       if (error) {
25         throw error
26       }
27       res.status(200).json(results.rows)
28     })
29   }
30 })
31
32 app.listen(8000, function () {
33   console.log("Server running");

```

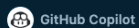
Detection & Remediation

- Prompt for insecure coding patterns
- Use GHAS Code Scanning results
- Write custom CodeQL queries
- Increase your knowledge of secure coding patterns
- Create custom Secret Scanning patterns



stebje

It looks like this code is insecure. Help me understand what the issue is and how to resolve it.



Yes, this code is vulnerable to SQL injection attacks. The issue is that the `search` parameter is directly concatenated into the SQL query string without any sanitization or validation. This means that an attacker could potentially inject malicious SQL code into the query, which could lead to data theft or destruction.

To fix this issue, you should use parameterized queries instead of concatenating strings. Parameterized queries use placeholders for the input values, which are then bound to the query separately. This ensures that the input values are properly sanitized and validated before being used in the query.

Here's an example of how to use parameterized queries with the `pg` library:

```
const search = req.params.q;

if (search !== '') {
  const sqquery = "SELECT * FROM users WHERE name = $1";
  const values = [search];
  pool.query(sqquery, values, (err, res) => {
    console.log(err, res);
    pool.end();
  });
}
```

In this example, the `$1` placeholder is used for the `search` parameter, and the actual value is passed in as an array of values to the `pool.query()` function. This ensures that the `search` parameter is properly sanitized and validated before being used in the query.

```
4
5 const pool = new Pool( config: {
6   user: 'postgres',
7   host: 'localhost',
8   database: 'postgres',
9   password: 'process.env.POSTGRES_PW',
10  port: 5432,
11 });
12
13 var app = express();
14 app.use(bodyParser.json());
15 app.use(bodyParser.urlencoded({ extended: true }));
16
17 app.get('/', function (req, res) {
18   const search = req.params.q;
19
20   if (search !== '') {
21     var sqquery = "SELECT * FROM users WHERE name = \"\" + search + \"\"";
22     pool.query( queryTextOrConfig: sqquery, callback: (err, res) => {
23       console.log( message: err, optionalParams[0]: res);
24       pool.end();
25     });
26   }
27 });
28
29 app.listen( port: 8000, callback: function () {
30   console.log( message: 'Example app listening on port 8000!');
31 });
```



Copilot + GHAS

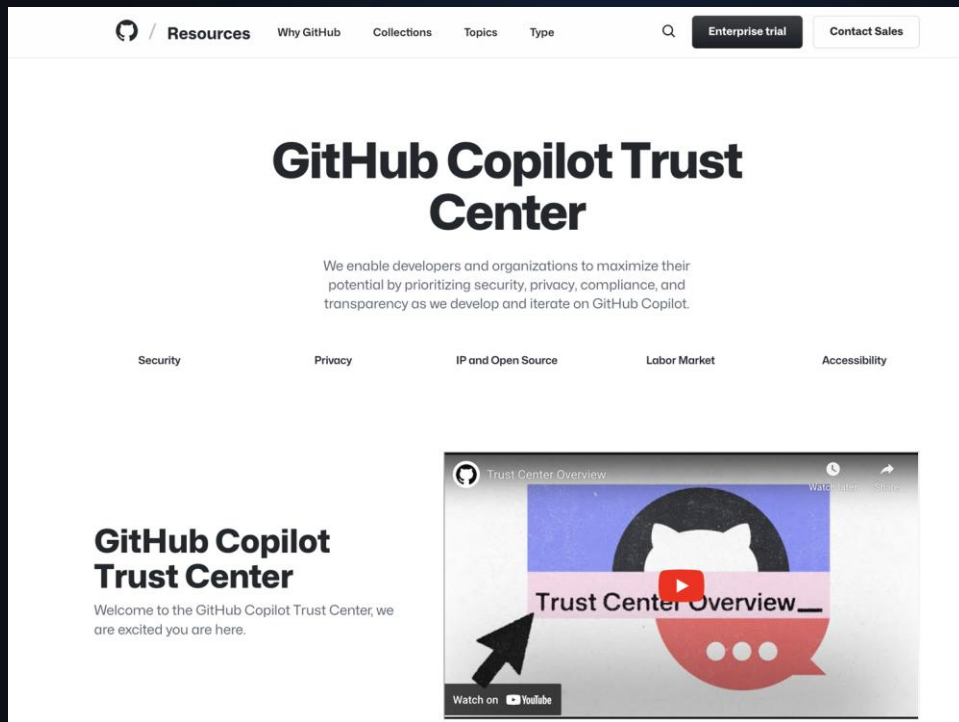
- Copilot is not a replacement of GHAS features
- Copilot can be used in tandem with GHAS features to detect and remediate vulnerabilities earlier during the SDLC
 - GHAS Code scanning results
 - GHAS Secret scanning



Security & Trust

Copilot Trust Center

- Security
- Privacy
- Data flow
- Copyright
- Labor market
- Accessibility
- Contracting



Wrap Up



Thank you

